AD-A088 079 KRANNERT GRADUATE SCHOOL OF MANAGEMENT LAFAYETTE IN F/G 12/2
A GENERALIZED DECISION SUPPORT SYSTEM USING PREDICATE CALCULUS --ETC(U)
JUL 80 R H BONCZEK, C W HOLSAPPLE DAAG29-79-C-0154
UNCLASSIFIED ARO-16231.5-EL NL

(18) ARO (19) 16231.5-EL

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER

2. GOVT ACCESSION NO. AD A088079

3. RECIPIENT'S CATALOG NUMBER (12)

4. TITLE (and Subtitle) (6)
A GENERALIZED DECISION SUPPORT SYSTEM USING PREDICATE CALCULUS AND NETWORK DATA BASE MANAGEMENT*

5. TYPE OF REPORT & PERIOD COVERED
Technical Rept.

6. PERFORMING ORG. REPORT NUMBER
Contract No. DA79C0154

7. AUTHOR(s) (10)
Robert H. Bonczek
Clyde W. Holsapple
Andrew B. Whinston

8. CONTRACT OR GRANT NUMBER(s) (15)
DAAG29-79-C-0154

9. PERFORMING ORGANIZATION NAME AND ADDRESS
Purdue University
Krannert Graduate School of Management
West Lafayette, IN 47907

10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS

11. CONTROLLING OFFICE NAME AND ADDRESS (11)
U.S. Army Research Office
Post Office Box 12211
Research Triangle Park, NC 27709

12. REPORT DATE
July 1980

13. NUMBER OF PAGES (12) 36

14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office)

15. SECURITY CLASS. (of this report)
unclassified

15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

LEVEL II

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

DTIC ELECTE AUG 15 1980 S D C

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

The view, opinions, and/or findings contained in this report are those of the authors and should not be construed as an official department of the Army position, policy, or decision, unless so designated by other documentation.

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

In view of the growing prominence of corporate modeling, an important area of research concerns techniques for facilitating the design and utilization of models. In this paper we show how first-order predicate calculus can be used as a language for formally stating modeling knowledge. Furthermore, knowledge stated in this manner can be subjected to the →next page

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE
S/N 0102-LF-014-6601

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

404200

80 8 14 319

AD A088079

DDC FILE COPY

resolution principle. The result is that application specific modeling knowledge need not be embedded in a computer program. Rather, it can be stored in a data base and utilized as needed by a problem processing system employing resolution techniques. Advantages of a decision support system taking an approach of this sort are considerable modeling flexibility, capacity for automating the model formulation and execution processes, and compatibility with a high-level user interface language.

"A GENERALIZED DECISION SUPPORT SYSTEM USING PREDICATE CALCULUS AND NETWORK DATA BASE MANAGEMENT"

*

Robert H. Bonczek
Assistant Professor of Management
Krannert Graduate School of Management
Purdue University

Clyde W. Holsapple
Assistant Professor of Business Administration
University of Illinois

Andrew B. Whinston
Professor of Management and Computer Science
Krannert Graduate School of Management
Purdue University

July, 1980

Supported in part by Army Contract number DA79C0154

* The authors would like to thank the referees for their helpful comments.

Abstract

In view of the growing prominence of corporate modeling, an important area of research concerns techniques for facilitating the design and utilization of models. In this paper we show how first-order predicate calculus can be used as a language for formally stating modeling knowledge. Furthermore, knowledge stated in this manner can be subjected to the resolution principle. The result is that application specific modeling knowledge need not be embedded in a computer program. Rather, it can be stored in a data base and utilized as needed by a problem processing system employing resolution techniques. Advantages of a decision support system taking an approach of this sort are considerable modeling flexibility capacity for automating the model formulation and execution processes, and compatibility with a high-level user interface language.

Operations research is concerned with the issues of model building and interfacing models with appropriate data in order to produce some expectations, facts, or beliefs for the support of decision making. First order predicate calculus can be used as a language for assisting in these activities, for capturing the knowledge of an OR expert as to how models can be correctly built and utilized. We shall examine the implications that this method of representing modeling knowledge has for the design of computer-based decision support systems.

A survey by Naylor and Schauland [15] has documented the importance and growth of corporate modeling. The three shortcomings of corporate models most frequently cited by users are inflexibility of the model, poor documentation, and excessive data input requirements. Users indicated that the three most desirable features for corporate modeling are sensitivity analysis, simple data base utilization and flexibile report generation. An objective of the predicate calculus method for representing modeling knowledge is the avoidance of these shortcomings and the accomodation of the desirable features.

On the basis of their survey, Naylor and Schauland maintain that the future of corporate modeling depends upon 1) the development of languages that are high-level and user-oriented; 2) the linkage of production planning models into overall corporate models; 3) the linkage of optimization models into overall corporate models; 4) the integration of financial, production and marketing models in a flexible, facile way; and 5) the linkage of corporate models with environmental models. The utilization of predicate calculus, as introduced in this paper, furnishes a method for handling the mechanics of inter-model linkages. The method is flexible in terms of adding or revising inter-model linkages. Moreover it can be incorporated into a computer-based decision support system in such a way that the system's users can direct model formulation and execution via a very high level

problem language. Note that the predicate calculus is not the high level language with which a user states a problem. It is the language for capturing modeling knowledge and for subsequently utilizing that knowledge within the system in the attempt to solve the user's problem.

We begin with a brief review of first order predicate calculus and the resolution principle. This is followed by an examination of a) the important issues involved in the treatment of models within decision support systems and b) a way in which predicate calculus may be used to address these issues. The proposed method is illustrated with an example. In conclusion, unresolved questions and topics of future research are identified.

1. Review of Predicate Calculus and Resolution

The predicate calculus is a general system of logic consisting of inference rules and a language for making statements about some domain of discourse. These statements may be viewed as assertions or axioms. Given a set of axioms, inference rules may be applied to these axioms to produce new statements which are logically valid deductions. The resolution principle is an inference method in predicate calculus which permits automatic deduction. Devised by Robinson [17], the resolution principle requires that all axioms be in what is called the clause form of first order predicate calculus. For complete descriptions of predicate calculus and resolution the reader should refer to [17] and Nilsson [16]; many details are omitted from the present review.

The predicate calculus language is given by its syntax, which specifies a) the symbols that are permitted in predicate calculus expressions and b) the ways for combining those symbols into valid expressions. The fundamental construct of the predicate calculus is the predicate symbol, conventionally represented by uppercase letters. Our primary concern here is with predicate symbols that occur with arguments (e.g., P(x), GT (a,b), or R(y,f(w))). A predicate symbol can be interpreted as indicating the property of its argument ("x has property P") or the nature of a relationship among its arguments ("a is Greater Than b").

There are three kinds of arguments: constants, variables and functions. Lower case letters at the beginning of the alphabet and numerals will be used to denote constants. Variables are represented by lower case letters at the end of the alphabet. Functions are given by f_i where $i \geq 1$. For a given application area, the set of all constants pertinent to that application is called the domain of discourse. Variables fill the customary role of representing one or more of the constants in the domain of discourse. If a function has n arguments, then that function maps an n-tuple

of members of the domain of discourse into some member of the domain of discourse. Constants and functions are called terms. If t_1, t_2, ..., t_m are terms and P is a predicate symbol then P (t_1, t_2 ..., t_m) is an atomic formula.

We can now examine how valid predicate calculus expressions are constructed from terms, variables, predicate symbols, and certain special symbols. Legitimate predicate calculus expressions (i.e., axioms) are called well-formed formulas (wffs). All atomic formulas are wffs. If α is a wff, then $\sim \alpha$ is a wff. If α and B are wffs then $\alpha \Rightarrow$ B is a wff, $\alpha \wedge$ B is a wff, and $\alpha \vee$ B is a wff. Before presenting other types of wffs we pause to discuss the semantics of the predicate calculus presented so far.

As stated above, constant symbols are used to specify the domain of discourse of a particular application area. A function specifies a mapping that is meaningful for the application area. A predicate symbol specifies a property or relationship that has a meaning within the application area. Note that the specific interpretation of a given constant, function, or predicate symbol is embedded in a knowledge of the application area and not in the predicate calculus. Using this knowledge about the application area we can give an interpretation of either T (true) or F (false) to any atomic formula. Then, utilizing the semantics of the various special symbols ($\Rightarrow$, $\sim$, $\vee$, $\wedge$) that appear in a wff, an interpretation of the wff can be given. That is, every wff can be assigned a value of T or F. The special symbols are defined in conformance with the usual mathematical notions of implication, negation, disjunction, and conjunction [16].

To this point no mention has been made of variables in connection with well-formed formulas. Variables, together with assocaited quantifiers, can be used to shorten certain kinds of long wffs. In order to state a wff that makes the same assertion about every one of the members of some domain of discourse, we could write out the conjunctive expression formed

from each individual assertion. This long and somewhat repetitive wff can be more concisely expressed by using a universally quantified variable. For instance, rather than writing $P(a_1) \wedge P(a_2) \wedge P(a_3) \wedge \ldots \wedge P(a_n) \ldots$ where $\{a_1, a_2, \ldots, a_n, \ldots\}$ is the domain of discourse, we can simply write $\forall x\, P(x)$. The symbol "$\forall$" is the universal quantifier and has the usual mathematical meaning. In this example, x is said to be a universally quantified variable.

Another type of quantifier is the existential quantifier ($\exists$) which also has the usual mathematical meaning. Expressions with existentially quantified variables may be used to concisely represent a disjunctive wff composed of the same assertion (wff) about every member of a domain of discourse. Using $\{a_1, a_2, \ldots, a_n, \ldots\}$ as the domain of discourse, the wff $Q(a_n,a_1) \vee Q(a_n,a_2) \vee \ldots \vee Q(a_n, a_n) \ldots$ can be simply restated as $\exists y\, Q(a_n, y)$. The earlier notion of a "term" can now be extended to include universally and existantially quantified variables, thereby extending the earlier notions of an atomic formula and a wff.

For a given application area (and therefore domain of discourse) the predicate calculus provides a means for expressing knowledge about that area in the guise of wffs. The set of all <u>stated</u> wffs having a T interpretation for an application area will be referred to as the <u>axiom set</u> for that application. When stating knowledge about an application area in terms of axioms it is vital that the axioms (wffs) be consistent with one another. For example, P(a) and $\sim$P(a) should not appear in the same collection of axioms to be used in a specific application area, since they cannot both have a T interpretation. Note that as new knowledge is acquired about an application area, new axioms can be included in the axiom set.

Given an axiom set, the question arises as to whether one can make logically valid inferences from that set. From explicitily stated axioms, how can one infer other axioms which are implicit in the axiom set? The

resolution principle of Robinson furnishes an inference method which can be applied to any axiom set. Briefly, resolution begins with a "target" axiom (often referred to as a theorem) and determines whether it logically follows from the axiom set. Before resolution procedures can be applied, the theorem and the axiom set must be put into clause form. That is, each wff must be converted into one (or possibly more) equivalent wffs that have the clause form. A simple algorithm to accomplish this conversion is given in [16] and is not repeated here. The algorithm has such effects as eliminating implications, existential quantifiers, conjunctions, and explict universal quantifiers. Each of the resultant wffs is a disjunction of literals, where a literal is an atomic formula or a negated atomic formula. This disjunction is called a clause and, unless otherwise indicated, we assume an axiom set to be in clause form. All variables appearing in a clause are understood to be universally quantified.

As an initial step in resolution, the theorem to be proven is negated. (Resolution will then provide a proof by contradiction.) One literal, L*, of this negated theorem is chosen as a basis for unification [17]. Unification is a procedure whereby two clauses are reduced to a single clause (called the resolvent) by making argument substitutions between the two parent clauses, such that one of these clauses contains a literal that is the negation of a literal in the other clause. These two literals (which form the basis for the unification), disappear from the resolvent. In resolution, the negated theorem is one parent clause and the other parent clause is chosen from the axiom set. This latter clause must contain a literal that allows it to be unified with the negated theorem, based upon L*.

Linear resolution is a recursive process, in which the resolvent of a unification becomes one parent clause in the succeeding unification. The other parent clause is a clause chosen from the axiom set, such that it can be unified on the basis of some selected literal L* appearing in the

preceding resolvent. If a null resolvent arises as the result of these successive unifications, then the theorem has been proven. That is, the "target" axiom can be inferred from (or is implicit in) the explictly stated axiom set. A property of resolution as applied to predicate calculus is that if a theorem is true a proof will be found in a finite number of applications of the resolution process.

For a complete, detailed description of the resolution principle see [17]. Several clearly presented examples can be found in Figure 1 displays the formalism that is used in this paper to illustrate a resolution process. Clauses C_1 and C_2 are parent clauses. Here, unification is based on the literal ~PD (z, 1978). Using the indicated substitutions (z replaces d and 1978 replaces yr), the last literal of C_2 is the negation of ~PD (z, 1978). Thus C_1 and C_2 can be unified, giving the resolvent C_3. In the next unification (if one is possible) C_3 becomes a parent clause.

Van Emden [21] has noted that algorithmic implementations of the resolution principle differ according to methods used for handling two choices that must be made when performing resolution. One choice involves which literal to use as a basis for unification. Having made this choice, there may be many clauses in the axiom set that can be used for unification. The second choice is whether these should be examined with a breadth-first or depth-first strategy and which clause should be treated first, second, etc. We shall return to these implementation issues at the end of this paper. In the meantime the major objective is to demonstrate how the predicate calculus and resolution principle can be used for the formal representation and automatic utilization of modeling knowledge as a decision support system enhancement.

2. The Inclusion of Modeling within a Decision Support System

The importance of a modular approach of model development and design for decision support systems is well-known. Kleijnen [13] has pointed out its significance from the standpoint of reducing developmental costs. The modular approach is also valuable in terms of the flexibility it affords (see Sprague and Watson [20], Seaberg and Seaberg [18]). The term "module" will be used here to denote a model that can be applied either on a stand-alone basis or in tandem with other modules to form a more comprehensive model. The existence of a pool of modules will serve as our starting point for the incorporation of modeling capabilities into a decision support system (DSS). The pertinent issue is the identification of (hopefully general) methods for coordinating and managing these modules within a DSS context. That is, how are modules "tied" together to form a model in a DSS?

As pointed out by Bonczek, et. al. [4], model usage within a DSS can be broadly classified as falling into one of the following categories:

1) the DSS user gives a procedural specification of how the model is constructed,
2) the DSS user invokes a predefined model by name, or
3) the DSS user states what data are desired and the DSS formulates a model(s) that can satisfy the request.

Although category 1 involves a relatively low-level language, requiring procedural description, it does afford considerable flexibility in terms of the models that can be specified. The second category is non procedural, but has limited flexibility. The user does not need to indicate how a model is constructed, but merely names one of the group of available models. Each of these models was predefined during the design of the DSS.

Present-day decision support systems fall into the first two categories [4] of user-formulated models or system designer-formulated models. The third category involves system-formulated models. User needs are stated

in a non-procedural manner. The degree of modeling flexibility offered by such a DSS is a function of model building knowledge available to the DSS. (Just as with the other two categories, the flexibility also depends on the extent of the pool of modules.) The manner in which application-specific modeling knowledge is incorporated into a DSS will determine whether that DSS can deal with only one application area or whether it is capable of being used in many application areas.

This may be understood by considering Figure 2, which shows a generic description of decision support systems (also see Bonczek, et. al. [3]). The description maintains that a DSS has the three indicated subsystems. The problem processing system (PPS) is the subsystem that mediates user needs, stated in a language system (LS), with the decision support system's knowledge system (KS) in order to provide an answer to the user. Now if the PPS code depends upon the particular models that are to be available to a user, then alterations or additions of models (e.g., due to a change in the application area to be supported) necessarily require changes in the PPS code. That is, the valid ways for interfacing modules are embedded in the PPS software. This design approach is typical in the OR/operations management software packages presently available. The result of embedding knowledge about models for a specific application area in a PPS is a considerable degree of inflexibility.

Can this application-specific modeling knowledge be separated from a PPS such that the PPS is general? That is, can we devise a PPS such that its code does not need to change in order to support decision makers in various application areas? The PPS should also be invariant to changes in the modeling knowledge within a specific application area. The flexibility afforded by such a system is significant from the standpoint of the previously cited shortcomings and desirable features vis-a-vis corporate modeling. Further comments on the importance of this generality appear in [3] and [13].

To prevent application-specific modeling knowledge from entering into the PPS code, we propose that the predicate calculus be used to express application-specific modeling knowledge and that the resultant axiom set be stored (along with the module pool) in the KS. For our present purposes a KS may be thought of as a data base management (DBMS) facility. Although DBMS are conventionally used for storing factual descriptive data they can also be used to store procedural data (i.e., program modules) as suggested by Bonczek, et. al. [1]. The techniques for storage (and retrieval) of modules and predicate calculus expressions in a data base are not discussed here.

When we speak of predicate calculus as a "language" that can be used to formally express modeling knowledge, this does not imply procedurality among axioms in an axiom set. The wffs are used to state facts about how modules may be used in conjunction with each other and with certain data in order to effect certain results. Many such facts or axioms are compiled for a particular application area. In essence, these formally stated modeling facts capture the knowledge of an expert (i.e., an operations research practicioner) about a particular application. Different applications (e.g., different corporations) would have different axiom sets and different module pools. But whatever the application, its axiom set is stored in the KS and acted upon by the PPS in order to formulate a complete model in response to a user's statement of a problem.

An axiom set can be viewed as a collection of fragments of knowledge about how to use modules with data in conducting various analyses. Given that this knowledge has been elicited from OR experts, it can be used to automatically construct a model(s) which, when executed, gives a desired report. This can be accomplished by implementing the resolution principle as a part of the PPS. The next section of this paper provides an example of how modeling knowledge, stated as wffs, can be used in conjunction with

the resolution principle to build a model (out of modules) and determine its data inputs.

3. Stating Modeling Knowledge in the Predicate Calculus

Recall that a predicate indicates a particular relationship among its arguments. For our present purposes it is convenient to introduce the notion of an "operator" predicate. Notice that a module formally specifies an operational relationship among its various inputs and its outputs. A predicate that is used to denote a module (i.e., an operational relationship) will be referred to as an operator predicate. Its arguments are the module inputs and outputs. Operator predicates will be distinguished from other predicates by underlining the operator's predicate symbol.

As another addition to the usual predicate calculus, the idea of preconditions will be used. In state space analysis [16] preconditions are simply conditions that must be satisfied prior to applying an operator; an operator transforms one state into another state. An analogous situation exists with modules, for a module transforms one "information state" into another "information state". Before a module can be executed to effect the transformation, certain preconditions must have been met. That is, all inputs to the module must be fully instantiated. Those arguments of an operator predicate, whose instantiations are prerequistes to the execution of the corresponding module, will be referred to as preconditions. They will be underlined in the examples that follow.

A final, but minor, departure from predicate calculus conventions presented earlier is that numerals appearing as arguments will be treated as constants. However, all other symbols (a, b, c, α, π, div, γ, etc.) appearing as arguments should be understood to be variables. Constants are instantiated arguments; variables are not; functional terms will not be used.

A simple example has been chosen to illustrate the foregoing ideas. Suppose that our module pool contains a regression module, a prediction module, an operator that finds dividends from profit and shares, and an operator that finds profit from revenues and expenses. These four modules

are respectively denoted by the predicate symbols REGRESS, PREDICT, O_1, O_2. Facts about how these can be used could be added to the KS as the DSS is being initially readied or some of it could be added later, as incremental increases in the system's modeling knowledge. Whatever the case, suppose that we (as operations researchers, not as users) currently want to add the fact that the regression and prediction modules can be used, together with past sales data and certain sales-indicators, to project the revenue in a certain year. This fact can be stated as shown in wff (1) of Figure 3. Although it is not explicitly stated in (1), all variables in this wff are universally quantified. The same two modules can again be used in tandem, but with different data, in order to project expenses in a certain year. This modeling fact is captured in wff (3).

A precise word description of modeling knowledge represented in (1) would be fairly lengthy. For instance, if X are sales indicators in years YR, and x are same types of sales indicators in year yr and Y are sales in years YR, and X, Y as inputs to a regression module produce the output B and B along with x are inputs to a prediction module that gives output r, then the revenue in year yr is r. Expression (1) is clearly superior to the word description from the standpoints of conciseness, unambiguous documentation, and amenability to computerized processing. The latter is particularly significant when efforts are made to combine and then utilize fragments of modeling knowledge. In the usual situation, wherein many fragments are available for use, the word form would not be workable for automated formulation. Even if the word form were only to be used during "manual" (i.e., human mental) formulation, the mental processing could become quite cumbersome.

Two additional fragments of modeling knowledge are given in wffs (2) and (4). The former states that if "π" is profit in "yr" and if "shares" is the assumed number of shares and if "d" is the output of O_1

given the inputs "π" and "shares", then "d" is the dividend in "yr". The latter states the if "e" is the expense in "yr" and "r" is revenue in "yr" and "p" is the result of operator O_2 where "e" and "r" are preconditions, then "p" is the profit in "yr". The clause form for each of these wffs is given in wffs (5) through (8) of Figure 3. These clause forms are what must actually be stored in the KS. The ordering of these four fragments of modeling knowledge is irrelevant for our present purposes.

Figure 4 displays several meta-axioms that we will have occasion to use. All are stated in clause form and they make up a portion of the axiom set. Meta-axioms establish the meaning or sense of predicates appearing in the axiom set by indicating the relationship between predicate arguments and the data item types that can appear in a user's request for data. For instance, DIVIDEND, YEAR, and SH are all data item types that the user can refer to when making a request using the LS.

To summarize, we have a method for formally representing modeling knowledge. This formal representation can be stored in a computer-based knowledge system. Fragments of modeling knowledge are specified by a modeling expert (e.g., operations researcher), not by the DSS user. Knowledge specified in this manner can readily be added to or deleted from the KS on a continuing basis by utilizing common data base management techniques. The result is a separation of application-specific modeling knowledge from the PPS, thereby allowing PPS generality in terms of the kinds of modeling supported. Notice that the axiom set serves as a clear documentation of legitimate model building and utilization techniques for a particular application area. For comments on the importance of modeling procedure documentation see [18]. Continuing the above example, we shall now see how a PPS with resolution capabilities can formulate a model which, when executed, will provide data requested by a DSS user.

4. Model Formulation by Resolution

Recall that resolution makes use of the axiom set plus a predicate calculus expression that states exactly what it is that we are to attempt to infer from the axiom set. We earlier referred to this expression as a target axiom or a theorem. Now the user of a DSS also has a "target" in mind when specifying the data that are desired in terms of the LS. If the system's user is an upper-level manager, it is advantageous to furnish a LS that is English-like. That is, a user should not be forced to learn predicate calculus in order to use the DSS. Bonczek, et.al.[2] have shown that expressions in an extant, English-like, non-procedural problem language can be readily converted into clause form expressions in the predicate calculus. Using the problem language the user might type: FIND DIVIDEND FOR YEAR = 1979 AND SH = 100. The corresponding wff is $\exists$ z DIVIDEND (z) $\wedge$ YEAR(1979) $\wedge$SH(100) and converting the negation of this wff to clause form gives

$\sim$DIVIDEND (z) v $\sim$YEAR (1979) v $\sim$SH (100).

This negated "theorem" is taken as a parent clause in the first step of resolution. The connection between data item types appearing in the user's request and the modeling knowledge supplied by a modeling expert is established by using meta-axioms during the resolution process. Figure 5 traces the application of the resolution principle to the above data request. Observe the last resolvent shown in Figure 5, consisting of (12) and (13). In our use of resolution for model formulation, we do not need to explicitly complete the resolution. However, the procedure undertaken upon reaching (12) and (13) is, in effect, a virtual completion of the resolution.

Expression (13) consists of those literals in the last resolvent that do not have operator predicates. As pointed out in [2] each of these literals corresponds a data retrieval query that can be executed against the KS data base. The reader should consult [2] for details of how this works. Briefly, the aforementioned correspondence is automatically

established with the aid of the meta-axioms. Execution of the resultant retrieval query makes available a list (i.e., 0,1 or more) of those instantiations of a variable which permit a unification to occur. For example, ~ EXP-IND (u, 1979) corresponds to a retrieval query to retrieve expense-indicators for the year 1979. If the data base contains such expense indicators (denote them by U_o, where U_o represents a constant) then the expression EXP-IND (U_o, 1979) is T. The act of retrieving the actual values U_o may be viewed as a virtual unification ($U_o \rightarrow U$) with the last resolvent of Figure 5. Having the values U_o available for use by a module is akin to the substitution of U_o for U in (12).

In this way all literals in (13) vanish, having been virtually resolved via data retrieval techniques. Now, if a data retrieval query gives a null answer then we cannot proceed, but must attempt some further explicit resolution. We shall return to this issue momentarily; suppose, for the time being, that all retrievals are successful. Then (13) is eliminated and we are left with (12) plus the data retrieved for variables Y, X, U, V, x, and u. All literals in (12) involve operator predicates, each having corresponding executable modules. However, before a given module can be executed its preconditions must be satisfied. That is, any underlined variable must be instantiated. For instance, before O_1 can be executed, π must be replaced by some constant.

From the earlier data retrieval, the variables Y, X, U, V, x, and u have been instantiated. If we let X_o and Y_o represent the instantiations of X and Y, then in (12) we have the literal ~ REGRESS (X_o, Y_o, B). If we next execute the regression module, then instantiations of B_o of B are obtained. As a result of this execution we can assert that REGRESS (X_o, Y_o, B_o) is T. Note that this unifies with ~REGRESS (X_o, Y_o, B) and results in the substitution of B_o for B. This means that the prediction module can be executed, with B_o and K_o as inputs, in order to eliminate ~PREDICT (B_o, X_o, r)

from (12) via a virtual unification. The virtual resolution of (12') to the null resolvent by an ordered succession of module executions is traced in Figure 6. As a starting point for this trace we take the virtual resolvent (12') resulting from the application of successful data retrievals. Symbols with a "o" subscript should be understood to represent constants that result from data retrieval or module execution.

Figure 6 gives an ordering of module executions (14-17) and the substitutions describe the necessary inter-module data linkages. The null resolvent is virtually obtained by appropriate module execution. Remember that the original user problem was to determine the 1979 dividend if there are 100 shares. Equivalently the problem was to find an instantiation (i.e., value) of z such that $\exists z$ DIVIDEND (z) $\wedge$ YEAR (1979) $\wedge$ SH (100). Let's review what the PPS has done to solve this problem. The PPS made use of the KS, which contained modeling knowledge (the axiom set), procedural knowledge (the module pool), and mundane descriptive knowledge (data on past sales and on sales-indicators). Explicit resolution was used (Figure 5) until a resolvent was obtained (12, 13) that could not be further unified with any axiom in the axiom set. Virtual resolution was performed in the guise of data retrieval, the details of which cannot be given here (see [2]). If all literals having non-operator predicates are eliminated by data retrieval, then further virtual resolution in the guise of ordered module execution is attempted. If a null resolvent is virtually attained then the answer to the user's request has been discovered (Z_o in the preceding example). Green's [10] answer tracing mechanism can be usefully applied, to keep track of where the answer is, in the face of many substitutions.

Thus the PPS has formulated a model consisting of several modules in response to a user-stated problem. The module ordering could be established as shown in Figure 6 or obtained more directly by a matching of preconditions with outputs appearing in (12'). Not only was a model

formulated by establishing data relationships among modules, but the PPS also automatically handled the data interfaces between modules and the data base. The entire process was directed by a user who does not need any expertise in predicate calculus or data base management techniques.

Notice that the approach to PPS design, as outlined in the two preceding paragraphs, is general. The processing methods proposed, and illustrated in the example, do not depend upon the specific example domain. In order to operate, the PPS does depend upon

- the existence of modules (not upon the nature of the modeling performed by each module)
- the existence of relevant data for the modules, organized according to either network data base techniques or a mixed knowledge representation method (as detailed in [2])
- the existence of formally stated modeling knowledge (as in Figure 3) pertinent to the available modules and data.

These three kinds of knowledge are stored in a generalized decision support system's KS, but they vary from one application domain to another. One application will involve different modules, data, and/or modeling facts than another application. Thus the contents of a decision support system's KS will differ from one application to the next and this enables the PPS to be invariant from one application to the next.

5. Extentions

The foregoing simple example was chosen for illustrative purposes. In practice, a much larger module pool, as well as a more extensive axiom set, would be available to the PPS. Thus a particular operator predicate may appear in many axioms and may be a participant in many of the models that the PPS can potentially construct. In practice some operator predicates may have more detailed or extensive argument sets than those in the previous example. For example, we may want to utilize the R^2 output of a regression module. Then (1) might become:

$$\underline{\text{REGRESS}}(\underline{Y}, \underline{X}, B, R^2) \wedge \underline{\text{PREDICT}}(\underline{B}, \underline{x}, r) \wedge \text{SALES}(Y, YR_1) \wedge \text{SALES-IND}(x, YR_1)$$
$$\wedge \text{SALES-IND}(x, yr) \wedge \text{GT}(R^2, .36) \Rightarrow \text{REV}(r, yr)$$

where GT is a "greater than" predicate. This axiom states that if $R^2 > .36$ then "r" will considered as revenue in "yr" otherwise it will not be taken as a valid estimation of revenue.

An interesting research issue is how to implement evaluative capabilities into the modeling knowledge. The use of logical comparisons (e.g., GT) is a step in this direction. There are several ostensible ways to treat comparison predicates: treat them as operator predicates, treat them as a special class of predicates whose operational details should be incorporated into the PPS, etc. The impact of permitting evaluative capabilities is that they allow branching knowledge to be given in the axiom set. (Of course, branching also exists within modules). For example,

$$\text{REGRESS}(\underline{Y}, \underline{X}, B, R^2) \wedge \text{LE}(R^2, .36) \wedge \text{ECON-DATA}(z, yr) \wedge \underline{\text{SALES-FORCAST}}(z, k)$$
$$\Rightarrow \text{REV}(k, yr)$$

could be added to the axiom set. If an $R^2 \leq .36$ is obtained then the model would involve some sort of sales-forcasting module (other than PREDICT) with appropriate economic data from the data base.

6. Implementing the Resolution Principle

Earlier the question was raised as to what happens if we reach a point where a data retrieval implied by some literal in the resolvent is not successful. The resolvent in question here is assumed to be incapable of further explicit resolution. That is, no attempts at virtual resolution via data retrieval are made until a resolvent is attained which cannot be unified with any of the clauses in the axiom set. A similar problem may be encountered even if all data retrievals are successful. It may be impossible to establish an ordering over the remaining operator predicates. Either type of impasse will necessitate some backtracking.

It is in this connection that the selection rule and search strategy used in implementing the resolution principle are important. Remember that these involve the two kinds of choices made in resolution implementations: selecting a literal to use as the basis of unification and a search strategy for exploring those members of the axiom set which can be unified on the basis of the selected literal. As the axiom set grows, the search tree for a given "theorem" can become very large. Van Emden [21] comments that the combinatorial problem in implementing the resolution principle has led to criticisms in terms of the practicality of resolution. But he proceeds to point out that nothing in the resolution principle requires an implementation of the "uniform" or "saturation" variety. Several implementations are cited that use various heuristics to guide the inference procedure In other implementations a foreknowledge of characteristics of an application area are exploited to provide a savings.

A number of techniques have been devised to reduce the quantity of extraneous and irrelevant resolvents that can arise during resolution, thereby reducing the amount of backtracking. Among these are semantic resolution [19], which uses ordered predicates to partition the clauses into classes within which no resolvents are formed; lock resolution [5], which indexes clause elements and only permits resolution on the predicates

with the smallest index; input resolution [6], in which for each resolvent one of the parent clauses is either an original axiom or the negated conclusion. The first two techniques are complete (they always find a solution if one exists), while the latter is not. However, input resolution is efficient and simple to implement.

The present authors are currently engaged in research aimed at the discovery of heuristics, that go beyond those noted above, to guide resolution in the case of model formulation. This research is taking several directions, that we shall briefly describe here. First, it appears that the meta-axioms can be used to considerably narrow the search tree. Secondly, the literals within a resolvent can be ordered (based on the degree of instantiation) as candidates for being chosen as a basis for the next unification. Third, unification on the basis of a literal that contains an operator predicate may be desirable to avoid in the early stages of resolution. Fourth, it may be useful to base a search strategy upon the presence or absence of operator predicates in clauses that can be chosen at a given stage of the resolution.

Fifth we may be able to take advantage of the special form of axioms that have been used to represent modeling knowledge. It happens that each clauses in the axiom set given earlier is a special type of predicate calculus expression called a Horn clause, (see Horn, [12]). A Horn clause is a clause containing no more than one positive literal. Kowalski [14] has successfully implemented a resolution system for Horn clauses. Furthermore, there is a marked correspondence between modeling knowledge stated in Horn clauses and the knowledge representation methods used in successful production systems such as the MYCIN system of Davis et. al. [7].

Although the PPS dynamics may be formally stated in terms of resolution,

it must be emphasized that the PPS outlined here involves a substantial departure from resolution problem solving systems. The distinction is not only in terms of the PPS emphasis on model formulation, but also in terms of its usage of data base management techniques in place of resolution. The PPS is not simply a resolution system.

In terms of its purpose, the PPS most closely resembles STRIPS [9], among the resolution-oriented systems. The PPS constructs (and then executes) a plan of analysis in terms of modules; STRIPS constructs a plan of actions, specified in terms of operators, to be taken by a robot [3]. We shall cite the major methodological differences between STRIPS and PPS. First, STRIPS uses theorem proving to determine whether a goal is met by the current state. If the goal is not met, then the addition and deletion lists of a permissible operator (an operator whose preconditions are satisfied by the current state) are used to generate a new current state, that is then checked against the goal, etc. When resolution shows that a current state meets the goal then the sequence of operators used to generate the current state are taken as the robot's plan of action. The PPS does not take the STRIPS state space approach. In the PPS, operators (modules) do not have addition and deletion lists, but they are incorporated into the problem solving process via Horn clauses.

A second major methodological difference (and this is the key difference) lies in the nature of knowledge representation. In STRIPS, as in all resolution systems, knowledge about the environment is specified in terms of the predicate calculus. For nontrivial applications, the amount of environmental knowledge that must be specified in predicate form escalates and renders the automatic deduction process practically intractable. In the PPS, none (or very little [2]) of the environmental knowledge is

specified in predicate form. In the earlier example, for instance, the knowledge about sales, expenses, sales indicators and expense indicators is knowledge about the decision support system's environment. An attempt to maintain such information in the form of hundreds (or more likely thousands) of needed clauses, and to then use those clauses during resolution, would be unworkable. Large volumes of these kinds of environmental facts are more appropriately handled with data base management techniques [2]. Indeed, data base techniques were designed for (and are today widely used for) just such large volumes of simple facts.

To summarize, the proposed PPS uses resolution for dealing with a relatively small volume of modeling knowledge specified in the Horn clause form and it uses data base management for dealing with relatively voluminous environmental knowledge that is stored according to an appropriate data base structure [2]. The results are necessarily a significantly shorter axiom list and thus a reduced level of searching during resolution.

Conclusion

The work presented here is intended as a contribution towards the integration of OR/MS with MIS/DSS as discussed by Edelman [8], Hoffman [11], and Vazsonyi [22]. Predicate calculus was presented as a formal language for capturing (and also documenting) the modeling knowledge of OR experts. Utilization of the predicate calculus in this manner permits application-specific modeling knowledge to be separated from the PPS code of a decision support system. The predicate calculus expressions can be stored in the KS of a DSS in order to take advantage of the ease of update afforded by data base management techniques.

The result is a general, flexible DSS which addresses, in part, the three corporate modeling shortcomings appearing in the introduction. It reduces modeling inflexibility. Some aid is provided in terms of documentation of modeling knowledge. Large data input requirements to modules are handled automatically by the DSS. The methods introduced in this paper also permit a DSS that possesses, to some degree, the desirable features of corporate modeling alluded to in the introduction. Sensitivity analysis can be incorporated not only within modules, but one can also compare the results of various modeling techniques (e.g. different forcasting methods can be incorporated into the axiom sets). Data base utilization by modules is accomplished automatically. Flexible report generation can be provided by a "sub-pool" of report generation modules that is subject to facile updating.

This paper has presented a basic skeleton of a methodology for automatic model formulation and utilization. Moreover, techniques for implementation were identified. Further research is needed in the realms of user language extentions, validation of the internal consistency of axiom sets, and resolution heuristics for model formulation.

References

[1] R. H. Bonczek, C. W. Holsapple and A. B. Whinston, "Design and Implementation of an Information Base for Decision Makers", Proceedings of the National Computer Conference, (1977).

[2] R. H. Bonczek, C. W. Holsapple and A. B. Whinston, "The Integration of Net rk Data Base Management and Problem Resolution", Information Systems, Vol. 4, 143-154 (1979).

[3] R. H. Bonczek, C. W. Holsapple and A. B. Whinston, "New Ideas in Decision Support", submitted for publication (1980).

[4] R. H. Bonczek, C. W. Holsapple and A. B. Whinston, "The Evolving Roles of Models within Decision Support Systems", Decision Sciences, Vol. 11 (1980).

[5] R. S. Boyer, "Locking: A Restriction of Resolution," Ph.D. Thesis, University of Texas, Austin, 1971.

[6] C. Chang and R. C. Lee, Symbolic Logic and Mechanical Theorem Proving, Academic Press, New York, 1973.

[7] R. Davis, B. Buchanan and E. Shortliffe, "Production Rules as a Representation for a Knowledge-Based Consultation Program", Artificial Intelligence, vol. 8, 15-45 (1977).

[8] F. Edelman, "They Went Thataway", Interfaces, vol. 7, no. 3, 39-43, (1977)

[9] R. E. Fikes and N. J. Nilsson, "STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving," Artificial Intelligence, vol. 2, 189-208 (1971).

[10] C. Green, "Application of Theorem Proving to Problem Solving", Proceedings of International Joint Conference on Artificial Intelligence (1969).

[11] G. M. Hoffman, "The Contribution of Management Science to Management Information", Interfaces, vol. 9, no. 1, 34-39, (1978).

[12] A. Horn, "On Sentences which are True of Direct Unions of Algebras", Journal of Symbolic Logic, vol. 16, 14-21, (1951).

[13] J. P. C. Kleijnen, "Computers and Operations Research: A Survey", Computers and Operations Research, vol. 3, 327-335 (1976).

[14] R. A. Kowalski, "Predicate Logic as a Programming Language", Proceedings of IFIP Conference (1974).

[15] T. H. Naylor and H. Schauland, "A Survey of Users of Corporate Planning Models", Management Science, Vol. 22, 927-937 (1976).

[16] N. Nilsson, Problem Solving Methods in Artificial Intelligence, McGraw-Hill, 1971.

[17] J. A. Robinson, "A Machine-Oriented Logic Based on the Resolution Principle", Journal of the ACM, vol. 12, 23-41, (1965).

[18] R. A. Seaberg and C. Seaberg, "Computer Based Decision Systems in Xerox Corporate Planning", Management Science, vol. 20, 575-584 (1973).

[19] J. R. Slagle, "Automatic Theorem Proving with Renamable and Semantic Resolution," Journal of the ACM, vol. 14, 687-697 (1967).

[20] R. H. Sprague and H. J. Watson, "A Decision Support System for Banks", OMEGA, vol. 4, 657-671 (1976).

[21] M. H. Van Emden, "Programming with Resolution Logic" in Machine Intelligence 8, (eds. E. W. Elcock and D. Michie), Halstead Press, New York, 1977.

[22] A. Vazsonyi, "Information Systems in Management Science", Interfaces, vol. 9, 72-77, (1978).

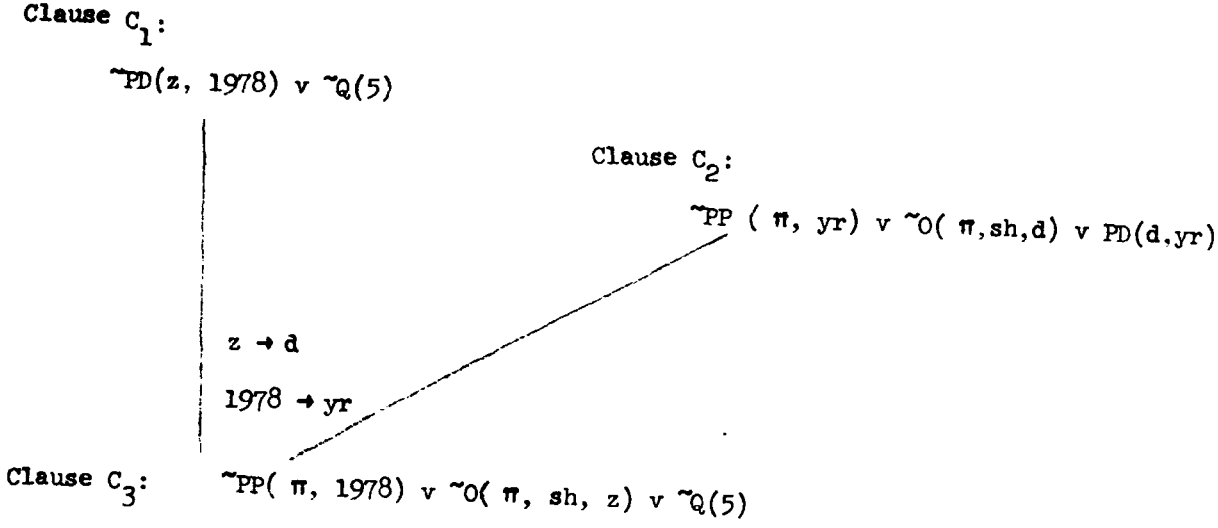

Figure 1. Resolution Process

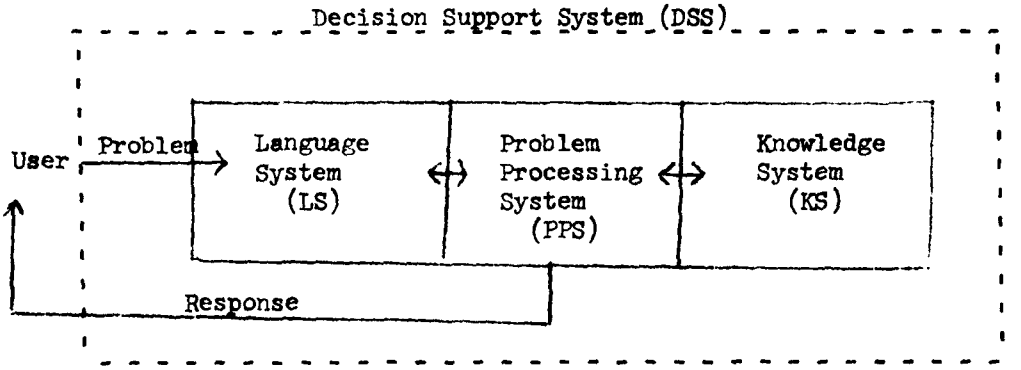

Figure 2. Generic Description of DSS

$$\underline{\text{REGRESS}}\ (\underline{Y}, \underline{X}, \beta) \wedge \underline{\text{PREDICT}}\ (\underline{\beta}, \underline{x}, r) \wedge \text{SALES}\ (Y, YR_1) \wedge \text{SALES-IND}\ (X, YR_1) \wedge \text{SALES-IND}\ (x, yr) \Rightarrow \text{REV}\ (r, yr) \quad (1)$$

$$\text{PR}\ (\pi, yr) \wedge \text{SH}\ (\text{shares}) \wedge O_1\ (\underline{\pi}, \underline{\text{shares}}, d) \Rightarrow \text{DIV}\ (d, yr) \quad (2)$$

$$\underline{\text{REGRESS}}\ (\underline{V}, \underline{U}, \gamma) \wedge \underline{\text{PREDICT}}\ (\underline{\gamma}, \underline{u}, e) \wedge \text{EXP}\ (V, YR_2) \wedge \text{EXP-IND}\ (U, YR_2) \wedge \text{EXP-IND}\ (u, yr) \Rightarrow \text{EXP}\ (e, yr) \quad (3)$$

$$\text{EXP}\ (e, yr) \wedge \text{REV}\ (r, yr) \wedge O_2\ (\underline{e}, \underline{r}, p) \Rightarrow \text{PR}\ (p, yr) \quad (4)$$

$$\sim\underline{\text{REGRESS}}\ (\underline{Y}, \underline{X}, \beta) \vee \sim\underline{\text{PREDICT}}\ (\underline{\beta}, \underline{x}, r) \vee \sim\text{SALES}\ (Y, YR_1) \vee \sim\text{SALES-IND}\ (X, YR_1) \vee \sim\text{SALES-IND}\ (x, yr) \vee \text{REV}(r, yr) \quad (5)$$

$$\sim\text{PR}\ (\pi, yr) \vee \sim\text{SH}\ (\text{shares}) \vee \sim O_1\ (\underline{\pi}, \underline{\text{shares}}, d) \vee \text{DIV}\ (d, yr) \quad (6)$$

$$\sim\underline{\text{REGRESS}}\ (\underline{V}, \underline{U}, \gamma) \vee \sim\underline{\text{PREDICT}}\ (\underline{\gamma}, \underline{u}, e) \vee \sim\text{EXP}\ (V, YR_2) \vee \sim\text{EXP-IND}\ (U, YR_2) \vee \sim\text{EXP-IND}\ (u, yr) \vee \text{EXP}\ (e, yr) \quad (7)$$

$$\sim\text{EXP}\ (e, yr) \vee \sim\text{REV}\ (r, yr) \vee \sim O_2\ (\underline{e}, \underline{r}, p) \vee \text{PR}\ (p, yr) \quad (8)$$

Figure 3. Selected Modeling Facts

~DIV (d, yr) v DIVIDEND (d) (9)

~DIV (d, yr) v YEAR (yr) (10)

~O_1 (π, sh, d) v SH (sh) (11)

Figure 4. Selected Meta Axioms (clause form)

~DIVIDEND (z) v ~YEAR(1979) v ~SH(100)

z → d

~DIV(d, yr) v DIVIDEND(d) (9)

~DIV(z, yr) v ~YEAR(1979) v ~SH(100)

1979 → yr
(z → d)

~DIV(d, yr) v YEAR (yr) (10)

~DIV(z,1979) v ~SH(100)

z → d
1979 → yr
(100 → shares)

~PR(π, yr) v ~SH(shares) v ~O_1(π,shares,d) v DIV(d,yr) (6)

~PR(π, 1979) v ~SH(100) v ~O_1(π, 100 z)

100 → sh
(z → d)

~O_1(π,sh,d) v SH (sh) (11)

~PR(π,1979) v ~O_1(π,100,z)

1979 → yr
π → p

~EXP(e,yr) v ~REV(r,yr) v ~O_2(e,r,p) v PR(p,yr) (8)

~O_1(π,100,z) v ~EXP(e,1979) v ~REV(r, 1979) v ~O_2(e,r,π)

1979 → yr

~REGRESS(Y,X,β) v ~PREDICT(β,x,r) v ~SALES(Y,YR_1) v ~SALES-IND(X,YR_1) v... (5)

~O_1(π,100,z) v ~EXP(e,1979) v ~O_2(e,r,π) v ~REGRESS(Y,X,β) v ~SALES(Y,YR_1) v ~SALES-IND(X,YR_1) v ~SALES-IND(x,1979)

1979 → yr

~REGRESS(V,U,γ) v ~PREDICT(γ,u,e) v ~EXP(V,YR_2) v ~EXP-IND(U,YR_2) v... (7)

~O_1(π,100,z) v ~O_2(e,r,π) v ~REGRESS(Y,X,β) v ~PREDICT(β,x,r) v ~REGRESS(V,U,γ) v ~PREDICT(γ,u,e)
v ~SALES(Y,YR_1) v ~SALES-IND(X,YR_1) v ~EXP(V,YR_2) v ~EXP-IND(U,YR_2) v ~SALES-IND(x,1979) v ~EXP-IND(u,1979) (12)

$$\sim O_1(\pi, 100, z) \vee \sim O_2(e, r, \pi) \vee \sim \text{REGRESS}(Y_o, X_o, \beta) \vee \sim \text{REGRESS}(V_o, U_o, \gamma) \vee \sim \text{PREDICT}(\beta, x_o, r) \vee \sim \text{PREDICT}(\gamma, U_o, e) \qquad (12')$$

$\beta_o \rightarrow \beta$
$\gamma_o \rightarrow \gamma$

$\{\text{REGRESS}(Y_o, X_o, \beta_o)$
$\text{REGRESS}(V_o, U_o, \gamma_o)\}$ resulting from executions of Regression module (14)

$$\sim O_1(\pi, 100, z) \vee \sim O_2(e, r, \pi) \vee \sim \text{PREDICT}(\beta_o, x_o, r) \vee \sim \text{PREDICT}(\gamma_o, u_o, e)$$

$r_o \rightarrow r$
$e_o \rightarrow e$

$\{\text{PREDICT}(\beta_o, x_o, r_o)$
$\text{PREDICT}(\gamma_o, u_o, e_o)\}$ resulting from execution of prediction module (15)

$$\sim O_1(\pi, 100, z) \vee \sim O_2(e_o, r_o, \pi)$$

$\pi_o \rightarrow \pi$

$\{O_2(e_o, r_o, \pi_o)\}$ resulting from the execution of module O_2 (16)

$$\sim O_1(\pi_o, 100, z)$$

$z_o \rightarrow z$

$\{O_1(\pi_o, 100, z_o)\}$ resulting from the execution of module O_1 (17)

null

Figure 6. The Virtual Resolution of (12') by Module Execution